# Agentsheets: A Tool for Building Domain-Oriented Visual Programming Environments

## (Demonstration)

Alex Repenning

Department of Computer Science and Institute of Cognitive Science
Campus Box 430
University of Colorado, Boulder CO 80309
(303) 492-1218, ralex@cs.colorado.edu
Fax: (303) 492-2844

**ABSTRACT**

Visual programming systems are supposed to simplify programming by capitalizing on innate human spatial reasoning skills. I argue that: (i) good visual programming environments should be oriented toward their application domains, and (ii) tools to build domain-oriented environments are needed because building such environments from scratch is very difficult. The demonstration illustrates how the visual programming system builder called Agentsheets addresses these issues and demonstrates several applications built using Agentsheets.

**PROBLEM**

Different approaches employed in visual programming systems vary in their degree of domain orientation:

- ***General purpose visual programming systems*** visually represent concepts found in conventional programming languages (e.g., boxes and arrows represent procedures and procedure calls). These systems have two advantages. First, they are applicable to a wide range of domains. Second, visual representations can facilitate syntactic constructions. For instance, shapes in BLOX Pascal (Figure 1) guide the programmer towards making syntactically correct constructions [3]. However, general purpose visual programming systems are difficult to use for non-programmers (e.g., BLOX Pascal still requires basic Pascal knowledge) and provide little - if any - gratification to expert programmers.

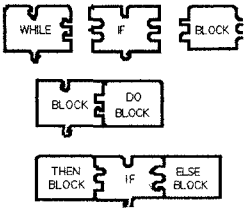


**Figure 1. BLOX Pascal**

- ***Domain-oriented visual programming systems*** represent artifacts pertinent to the end users' task at hand. A system like the pinball construction kit (Figure 2) is easy to use for non-programmers [1]. The process of programming in a construction kit-like environment consists of selecting components (e.g., pinball components) and assembling them. However, it is difficult to reuse systems tailored to a specific domain for other domains. For instance, the pinball construction kit could not be reused for numerical applications such as tax forms. Perhaps the biggest problem with domain-oriented systems is that no high-level substrate exists facilitating their creation.

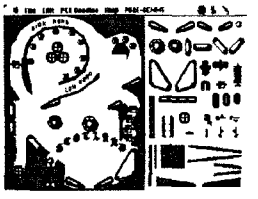

**Figure 2. Pinball Construction Kit**

Empirical tests suggest that despite their versatility, general purpose visual programming languages, in many situations, provide little or no advantage over conventional textual programming languages [4]. Domain-oriented visual programming systems, on the other hand, seem to be very useful for non-programmers but are hard to build from scratch [2]. The problem is, therefore, to find a new mechanism that will overcome the disadvantages of each approach without sacrificing their advantages.

**THE AGENTSHEETS SYSTEM**

Agentsheets facilitates the *creation* of domain-oriented visual programming systems. A spatial metaphor consisting of "communicating agents organized in grids" is used by a visual programming system designer to create a visual language specific to a problem domain.

In a typical utilization of Agentsheets, a visual programming system designer will define the look and behavior of domain-specific building blocks called agents. The behaviors of agents determine the meaning of spatial arrangements of agents (e.g., what does it mean when two agents are adjacent to each other) as well as the reaction of agents to user events (e.g., how does an agent react if it gets activated by the user). These agents constitute the elements of a high-level, domain-oriented visual programming language that can be used readily by end users.

End users arrange agents in a work area. The work area has an underlying grid structure analogous to the rows and columns in a spreadsheet. Relationships between agents can

be explicitly specified by connecting agents with links or implicitly specified simply by position within the grid structure.

The Agentsheets system is object-oriented. It provides a large set of built-in agent classes supplying typical behaviors to visual programming system designers. The designer extends this set with new classes achieving the domain-oriented behavior.

## EXAMPLE APPLICATIONS

The spatial organization of agents in a grid and their ability to communicate with each other can be used to simulate the semantics of flow. Individual agents representing flow conductors (e.g., pieces of wire or water pipes) are connected simply by placing them next to each other. Figure 3 shows two applications relying on flow semantics. Users can interact with more complex conductor agents like switches and valves to control the flow.

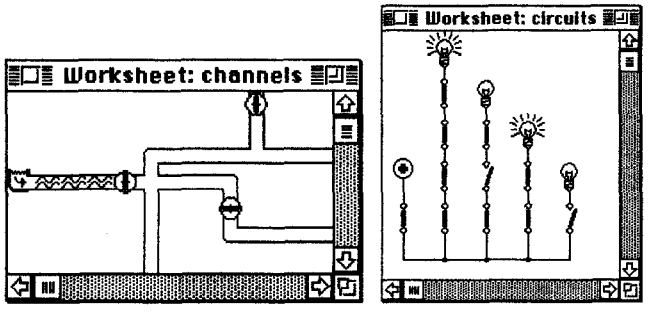


**Figure 3. Flow Semantics: Channels (left), Circuits (right)**

The ability to animate agents (change look, move, play sounds) can be used to realize very different Agentsheets applications. Figure 4 shows an application to simulate ecosystems. End users build environments consisting of mountains, grass tundra, etc. Then, they set out animals (wolfs, bear, etc.) in those environments.

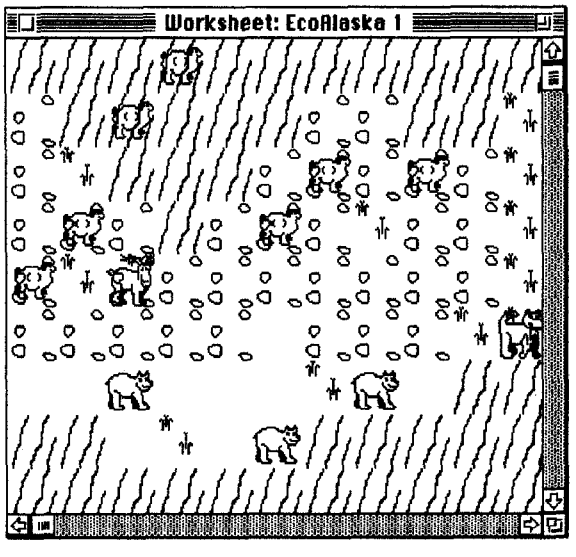


**Figure 4. Ecosystem Simulation**

The individual patches of the environment and the animals are agents. End users define simple behaviors for all ecosystem agents. The objective for users is to create ecologically stable environments as well as to get an intuition for the relationships between local behavior and global behavior.

Figure 5 below depicts a more "traditional" visual programming environment based on Agentsheets used to design and run phone-based voice dialog applications [5]. This application includes two different interfaces for two types of users:

- ***the customer interface*** simulates the very limited touch-tone-button-input/voice-output user interface of an ordinary telephone.
- ***the voice dialog designer interface*** shows a voice dialog designer a trace of usage and allows the designer to modify a design, while the system is in use, based on customer's suggestions or problems.

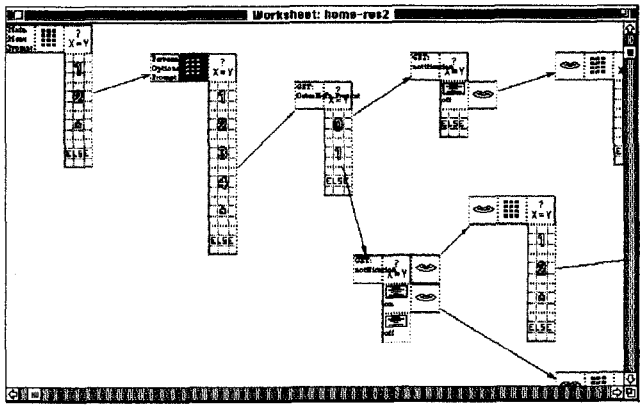


**Figure 5. Phone Based Voice Dialog Environment**

## REFERENCES

1. G. Fischer and A. C. Lemke, "Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication," *HCI*, Vol. 3, pp. 179-222, 1988.
2. E. P. Glinert, M. M. Blattner and C. J. Freking, "Visual Tools and Languages: Directions for the '90s," 1991 IEEE Workshop on Visual Languages, Kobe, Japan, 1991, pp. 89-95.
3. E. P. Glinert, "Towards "Second Generation" Interactive, Graphical Programming Environments," *IEEE Computer Society, Workshop on Visual Languages*, Dallas, 1986, pp. 61-70.
4. T. R. G. Green, M. Petre and R. K. E. Bellamy, "Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against the 'Match-Mismatch' Conjecture," *Empirical Studies of Programmers: Fourth Workshop*, New Brunswick, NJ, 1991, pp. 121-146.
5. A. Repenning and T. Sumner, "Using Agentsheets to Create a Voice Dialog Design Environment," *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, Kansas City, 1992, pp. 1199-1207.